DOI:
10.1007/978-3-030-25027-0_5

Document Version
Peer reviewed version

Nominal Syntax with Atom Substitutions: Matching, Unification, Rewriting

Jesús Domínguez and Maribel Fernández

Department of Informatics, King's College London, UK
jesus.dominguez@kcl.ac.uk, maribel.fernandez@kcl.ac.uk

Abstract. Unification and matching algorithms are essential components of logic and functional programming languages and theorem provers. Nominal extensions have been developed to deal with syntax involving binding operators: nominal unification takes into account α-equivalence; however, it does not take into account non-capturing substitutions, which are not primitive in nominal syntax. We consider an extension of nominal syntax with non-capturing substitutions and show that matching is decidable and finitary but unification is undecidable. We provide a matching algorithm and characterise problems for which matching is unitary, giving rise to expressive and efficient rewriting systems.

Keywords: nominal syntax, non-capturing substitution, rewriting, unification

1 Introduction

Nominal syntax is a generalisation of first-order syntax that deals with variable binding using atom permutations and freshness constraints (see [17, 28]). Nominal syntax uses two kinds of variables: atoms $a, b, \ldots$, which can be abstracted but not substituted ($[a]t$ means that a is abstracted in t), and meta-variables $X, Y, \ldots$, called simply variables, which may be decorated with atom permutations. Unification of nominal terms (i.e., modulo α-equivalence) is *decidable and unitary* [28]. Efficient algorithms exist that solve nominal unification problems in polynomial time [5, 21, 7]. Nominal matching (a form of unification where only one of the terms can be instantiated) can be solved in linear time [6].

Nominal unification and matching have applications in logic and functional languages [8, 25, 26, 2] and automated reasoning [13, 23, 10, 16, 11, 27] among others. However, nominal terms do not provide a built-in form of substitution for atoms that would permit direct definitions of systems such as the λ-calculus. Instead, atom substitution has to be defined explicitly, by rewrite rules or equations [15, 14], as in the following system, where (explicit) substitutions are sugared to $t\{a \mapsto t'\}$ and $a \# t$ means that a is not free in t.

$$
\begin{array}{lllll}
(\mathsf{Beta}) & & \mathtt{app}(\lambda[a]X, X') & \to X\{a \mapsto X'\} \\
(\sigma_{\mathtt{var}}) & & a\{a \mapsto X\} & \to X \\
(\sigma_{\epsilon}) & a \# Y \vdash & Y\{a \mapsto X\} & \to Y \\
(\sigma_{\mathtt{app}}) & & \mathtt{app}(X, X')\{a \mapsto Y\} & \to \mathtt{app}(X\{a \mapsto Y\}, X'\{a \mapsto Y\}) \\
(\sigma_{\mathtt{lam}}) & b \# Y \vdash & (\lambda[b]X)\{a \mapsto Y\} & \to \lambda[b](X\{a \mapsto Y\})
\end{array}
$$

An extension of nominal syntax with a primitive capture-avoiding atom substitution, which avoids the need to introduce explicit substitution rules, was presented in [12]; however, its rewriting theory was not developed. Here we show that unification in this extended syntax is undecidable in general but matching remains decidable (albeit no longer unitary) and the rewriting relation can be effectively computed. The undecidability result is obtained by reducing Hilbert's tenth problem to extended nominal unification, inspired by Goldfarb's proof of undecidability of second-order unification [18]. Our main contributions are an algorithm that computes complete sets of solutions for solvable matching problems, and a characterisation of a wide class of problems for which matching is unitary, inducing a well-behaved rewriting relation. This class includes the Beta and Eta reduction rules of the λ-calculus (we give details in Section 5). These results open the way for the development of expressive reasoning frameworks based on nominal syntax.

Related Work Our syntax for extended nominal terms is inspired by [12], where a dependent type system for extended terms is presented. Matching was used in [12] to type-check terms given a set of declarations for function symbols. It was noted that restrictions were needed to ensure unitary matching, however, no matching algorithm was provided. Capture-avoiding atom substitution was previously studied in the context of nominal algebra by Gabbay and Mathijssen [16, 15], but its unification theory was not considered.

In [13], a nominal reduction system for the λ-calculus is given, with an explicit atom-substitution operation defined by a set of rewrite rules. The extended nominal syntax proposed here reduces the verbosity of such systems by internalising capture-avoiding substitutions.

Efficient nominal unification algorithms were developed by Calvès and Fernández [5, 4] and Levy and Villaret [21]. Both approaches were later unified by Calvès [3]. Kumar and Norrish [19] also studied efficient forms of nominal unification. Cheney [9] proved that a more general version of nominal unification, called equivariant unification, is NP-complete.

We followed Goldfarb's methodology [18] to prove the undecidability of nominal unification extended with atom substitutions. Goldfarb [18] proved that second-order unification is undecidable by reducing Hilbert's tenth problem to a second-order unification problem. An alternative undecidability proof for second-order unification by a direct encoding of the Halting problem is given by Levy and Veanes [20], which could also be adapted to our language.

2 Background

Fix countably infinite, pairwise disjoint sets of **atoms**, $a, b, c, \ldots \in \mathcal{A}$; **variables**, $X, Y, Z, \ldots \in \mathcal{X}$; and **term-formers** $f, g, \ldots \in \mathcal{F}$. A **permutation** π is a bijection on a finite subset of $\mathcal{A}$ called **support** of π, $\mathcal{S}upport(\pi)$. A **swapping** $(a\ b)$ is a particular case where a maps to b, b maps to a and all other atoms c map to themselves. We follow the **permutative convention** [15, Convention 2.3] for

atoms throughout the paper, i.e., atoms a, b, c range permutatively over $\mathcal{A}$ so that they are always distinct unless stated otherwise. **Atom substitutions** ϕ, or just **a-substitutions**, are mappings with finite **domain** from atoms to terms, i.e., the set of atoms such that $\phi(a) \neq a$, written $\mathcal{D}om(\phi)$, is finite. Permutations π, a-substitutions ϕ, and **terms with atom substitutions** s, t, or just **(extended) terms**, are generated by the following grammar.

Definition 1 (Syntax).

$$\pi ::= \mathsf{Id} \mid \pi(a\ b) \quad \phi ::= \mathsf{Id} \mid [a \mapsto s]\phi \quad s, t ::= a \mid \phi\hat{}\pi{\cdot}X \mid [a]s \mid fs \mid (s_1, \ldots, s_n)$$

The final Id is usually omitted from permutations and a-substitutions. Write π^{-1} for the **inverse** of π, e.g., if $\pi = (a\ b)(b\ c)$ then $\pi(c) = a$ and $c = \pi^{-1}(a)$. A-substitutions are *simultaneous* bindings, abbreviated as $[a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]$ where atoms a_i are pairwise distinct. Write $\phi^{-a_1,\ldots,a_n}$ for the a-substitution ϕ with domain restricted to $\mathcal{D}om(\phi)\backslash\{a_1, \ldots, a_n\}$. $\mathcal{I}mg(\phi)$ denotes the set of terms $\{\phi(a) \mid a \in \mathcal{D}om(\phi)\}$. Term constructors as given in Def. 1 are called respectively **atoms**, **moderated variables**, **abstractions**, **function applications** (where $f()$ is denoted as f) and **tuples** ($n \geq 0$). A moderated variable $\phi\hat{}\pi{\cdot}X$ comprises a variable X, and **suspended** permutation π and a-substitution ϕ. As in first-order syntax, variables denote unknown parts of the term, but here they are decorated with permutations and atom-substitutions, that will act when the variable is instantiated, as shown below. We abbreviate $\mathsf{Id}\hat{}\pi{\cdot}X$ (resp. $\phi\hat{}\mathsf{Id}{\cdot}X$) as $\pi{\cdot}X$ (resp. $\phi{\cdot}X$) and $\mathsf{Id}\hat{}\mathsf{Id}{\cdot}X$ as X if there is no ambiguity.

Permutations **act** on terms and a-substitutions; $\circ$ denotes composition:

$$\pi{\cdot}a \triangleq \pi(a) \quad \pi{\cdot}[a]t \triangleq [\pi(a)]\pi{\cdot}t \qquad \pi{\cdot}ft \triangleq f\pi{\cdot}t \quad \pi{\cdot}(t_1, \ldots, t_n) \triangleq (\pi{\cdot}t_1, \ldots, \pi{\cdot}t_n)$$
$$\pi{\cdot}(\phi\hat{}\pi'{\cdot}X) \triangleq (\pi{\cdot}\phi)\hat{}(\pi \circ \pi'){\cdot}X \text{ where } \pi{\cdot}\mathsf{Id} \triangleq \mathsf{Id},\ \pi{\cdot}([a \mapsto t]\phi) \triangleq [\pi(a) \mapsto \pi{\cdot}t](\pi{\cdot}\phi)$$

Write $V(t)$ for the set of variable symbols appearing in a term t and $A(t)$ for the set of atoms in t; this includes atoms in the domain and image of a-substitutions and atoms in the support of permutations.

A **position** p, q is a string of positive integers denoting a path in the abstract syntax tree of a term. The set of **positions** of a term s, $\mathcal{P}os(s)$, is defined inductively as usual [1] with an additional case for a moderated variable:

$$\mathcal{P}os([a_1 \mapsto t_1; \cdots; a_n \mapsto t_n]\hat{}\pi{\cdot}X) \triangleq \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i \cdot p \mid p \in \mathcal{P}os(t_i)\}.$$

An arbitrary ordering (e.g., lexicographic) is chosen when defining the positioning of the terms in the image of suspended a-substitutions. Since we are dealing with simultaneous a-substitutions, the choice of ordering does not matter. The size of a term t, $|t|$, is the cardinality of $\mathcal{P}os(t)$. Call $t|_p$ the **subterm** of t at position p. If $p \in \mathcal{P}os(t)$, then $t[s]_p$ denotes the term obtained from t by replacing its subterm at position p by the term s and $(\cdots(s[t_1]_{p_1})\cdots)[t_n]_{p_n}$ is abbreviated as $s[t_1 \cdots t_n]_{p_1 \cdots p_n}$.

Example 1. Let $\mathtt{map}, \mathtt{cons}$ and $\mathtt{nil}$ be term-formers; $\mathtt{map}([a]F, \mathtt{cons}(H, \mathtt{nil}))$ is a term and so is t defined as $\mathtt{cons}([a \mapsto H]{\cdot}F, \mathtt{map}([b]F, \mathtt{nil}))$. $A(t) = \{a, b\}, V(t) =$

$\{F, H\}$. $\mathcal{P}os(t) = \{\epsilon, 1, 11, 111, 12, 121, 1211, 12111, 1212\}$ so that, $t|_{111} = H$ and $t|_{1212} = \mathtt{nil}$, for instance. See [28, 13, 12] for more examples.

Call $a \# t$ a **freshness constraint**. Let $\Delta, \nabla, \ldots$ range over finite sets of **primitive constraints** of the form $a \# X$; call such sets **freshness contexts**. Call $s \approx_\alpha t$ an **α-equivalence constraint**. Write $\nabla \vdash a \# t$ and $\nabla \vdash s \approx_\alpha t$, called **freshness** and **α-equivalence judgements** respectively, when a derivation exists using the syntax-directed rules from Def. 2 where, for a-substitutions ϕ, ϕ' and permutations π, π', $\mathcal{D}om(\phi) \cup \mathcal{D}om(\phi')$ is abbreviated as $\mathcal{D}omP(\phi, \phi')$ and $\mathcal{S}upport(\pi) \cup \mathcal{S}upport(\pi')$ as $\mathcal{S}upportP(\pi, \pi')$. We write $a, b \# t$ (resp. $a \# s, t$) instead of $a \# t, b \# t$ (resp. $a \# s, a \# t$), and abbreviate $\varnothing \vdash s \approx_\alpha t$ as $s \approx_\alpha t$.

Definition 2 (Freshness and α-equivalence judgements).

$$\frac{}{\nabla \vdash a \# b}\ (\#\mathbf{ab}) \qquad \frac{}{\nabla \vdash a \# [a]s}\ (\#[\mathbf{a}]) \qquad \frac{\nabla \vdash a \# s}{\nabla \vdash a \# [b]s}\ (\#[\mathbf{b}]) \qquad \frac{\nabla \vdash a \# s}{\nabla \vdash a \# fs}\ (\#\mathbf{f})$$

$$\frac{\bigwedge_{b \in \mathcal{D}om(\phi) \cup \{a\}} (\nabla \vdash a \# \phi(b) \vee (\pi^{-1}(b) \# X \in \nabla))}{\nabla \vdash a \# \phi\hat{}\pi \cdot X}\ (\#\mathbf{X}) \qquad \frac{\nabla \vdash a \# s_1 \cdots \nabla \vdash a \# s_n}{\nabla \vdash a \# (s_1, \ldots, s_n)}\ (\#\mathbf{tupl})$$

$$\frac{}{\nabla \vdash a \approx_\alpha a}\ (\approx_\alpha \mathbf{a}) \qquad \frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash [a]s \approx_\alpha [a]t}\ (\approx_\alpha [\mathbf{a}]) \qquad \frac{\nabla \vdash (b\ a) \cdot s \approx_\alpha t \quad \nabla \vdash b \# s}{\nabla \vdash [a]s \approx_\alpha [b]t}\ (\approx_\alpha [\mathbf{b}])$$

$$\frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash fs \approx_\alpha ft}\ (\approx_\alpha \mathbf{f}) \qquad \frac{\nabla \vdash s_1 \approx_\alpha t_1 \ \cdots \ \nabla \vdash s_n \approx_\alpha t_n}{\nabla \vdash (s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)}\ (\approx_\alpha \mathbf{tupl})$$

$$\frac{\bigwedge_{a \in (\mathcal{D}omP(\phi, \phi') \cup \mathcal{S}upportP(\pi, \pi'))} (\nabla \vdash \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)) \vee (a \# X \in \nabla))}{\nabla \vdash \phi\hat{}\pi \cdot X \approx_\alpha \phi'\hat{}\pi' \cdot X}\ (\approx_\alpha \mathbf{X})$$

The most interesting rules are $(\#\mathbf{X})$ and $(\approx_\alpha \mathbf{X})$. The first one specifies that a is fresh in $\phi\hat{}\pi \cdot X$ if it is fresh in the image by ϕ of any atom that could occur in an instance of $\pi \cdot X$. The second ensures that the atom actions produce the same effect for any valid instance of X, in other words, any atom that could be affected by the atom actions suspended in X is either affected in the same way on both sides of the equality constraint, or it must be fresh in X. The relation $\approx_\alpha$ is indeed an equivalence relation [12].

Example 2. We can derive $a \# [b \mapsto Y]\hat{}(a\ c) \cdot X$ from $\nabla_1 = \{a \# Y, c \# X\}$ or from $\nabla_2 = \{b \# X, c \# X\}$ using rule $(\#\mathbf{X})$, and $[c \mapsto (a\ b) \cdot Y] \cdot X \approx_\alpha [b \mapsto Y] \cdot (b\ c) \cdot X$ from $\nabla_1 = \{b \# X, a \# Y, b \# Y\}$ or $\nabla_2 = \{b \# X, c \# X\}$ using rule $(\approx_\alpha \mathbf{X})$. In contrast, using standard (non-extended) nominal syntax, for each derivable constraint there exists a unique least freshness context entailing it [28].

The action of an a-substitution ϕ on a term t relies on a freshness context ∇ and therefore is defined over **terms-in-context**, written $\nabla \vdash t$, or simply $\vdash t$ if $\nabla = \varnothing$. Below we abbreviate $(\nabla \vdash t)\phi$ as $\nabla \vdash t\phi$.

Definition 3 (A-substitution action).

$$\nabla \vdash a\phi \triangleq \nabla \vdash \phi(a) \quad \nabla \vdash (ft)\phi \triangleq \nabla \vdash ft\phi$$
$$\nabla \vdash (t_1, \ldots, t_n)\phi \triangleq \nabla \vdash (t_1\phi, \ldots, t_n\phi)$$
$$\nabla \vdash (\phi'\hat{}\pi{\cdot}X)\phi \triangleq \nabla \vdash (\phi' \bullet \phi)\hat{}\pi{\cdot}X \quad \textit{where } \bullet \textit{ denotes composition}$$
$$\nabla \vdash ([a]t)\phi \triangleq \nabla \vdash [b]((a\ b){\cdot}t)\phi^{-b} \quad \textit{where } \nabla \vdash b \# t, \mathcal{I}mg(\phi)$$

A-substitutions work uniformly on α-equivalence classes of terms, that is, the choice of b in Definition 3 is irrelevant [12]. Capture-avoidance is guaranteed by selecting an α-equivalent representative of $\nabla \vdash [a]t$, i.e., $\nabla \vdash [b](a\ b){\cdot}t$, with fresh b. There exists always some $b \in (\mathcal{A} \setminus (A(t) \cup A(\mathcal{I}mg(\phi))))$ such that $\nabla \vdash b \# t, \mathcal{I}mg(\phi)$, assuming primitive constraints $b \# X$ in ∇ for each X in $(V(t) \cup V(\mathcal{I}mg(\phi)))$, since variables have finite support [24]. We assume ∇ is large enough (in practice, it can be augmented whenever required). This approach is also taken in [8, 13, 12] and tacitly assumed in the rest of the paper.

Variable substitutions $\sigma, \theta, \ldots$, or just **v-substitutions**, are mappings from variables to terms, with finite **domain** $\mathcal{D}om(\sigma)$. They are generated by the grammar: $\sigma, \theta ::= \mathsf{Id} \mid [X \mapsto s]\sigma$ where Id is commonly omitted, and interpreted as *simultaneous* bindings, abbreviated $[X_1 \mapsto s_1; \ldots; X_n \mapsto s_n]$ where variables X_i are pairwise distinct. The application of a v-substitution θ to a moderated variable $\phi\hat{}\pi{\cdot}X$ induces the action of ϕ on the term $\pi{\cdot}\theta(X)$. The action of v-substitutions, σ, on terms, t, written $t\sigma$, is also parameterised by freshness contexts but left implicit in Def. 4. Given v-substitution σ and freshness contexts ∇, Δ, we write $\Delta \vdash \nabla\sigma$ to denote $\Delta \vdash a \# \sigma(X)$ for each $a \# X \in \nabla$.

Definition 4 (V-substitution action).

$$a\sigma \triangleq a \qquad ([a]t)\sigma \triangleq [a]t\sigma \quad (ft)\sigma \triangleq ft\sigma \quad (t_1, \ldots, t_n)\sigma \triangleq (t_1\sigma, \ldots, t_n\sigma)$$
$$(\phi\hat{}\pi{\cdot}X)\sigma \triangleq (\pi{\cdot}\sigma(X))(\phi\sigma) \quad \textit{where } \mathsf{Id}\sigma \triangleq \mathsf{Id} \quad \textit{and } ([a \mapsto s]\phi)\sigma \triangleq [a \mapsto s\sigma](\phi\sigma)$$

Permutations and a-substitutions commute: $\nabla \vdash \pi{\cdot}(s\phi) \approx_\alpha (\pi{\cdot}s)(\pi{\cdot}\phi)$ and $\nabla \vdash (\pi{\cdot}s)\phi \approx_\alpha \pi{\cdot}(s(\pi^{-1}{\cdot}\phi))$. Also, v-substitutions commute with permutations, $\nabla \vdash \pi{\cdot}(s\sigma) \approx_\alpha (\pi{\cdot}s)\sigma$, and a-substitutions, $\nabla \vdash (s\sigma)\phi\sigma \approx_\alpha (s\phi)\sigma$.

3 Unification, Matching and Rewriting

Definition 5. *Let C range over freshness and α-equality constraints. A* ***unification problem*** $\mathbf{P}$ *is a finite set of such constraints, where α-equivalence constraints are written as* ***unification constraints*** $s \stackrel{?}{\approx}_? t$. *A* ***solution*** *to* $\mathbf{P}$ *is a pair* $(\mathbf{F}, \sigma)$ *of a non-empty collection* $\mathbf{F}$ *of freshness contexts and a v-substitution* σ *such that* $\Delta \vdash C\sigma$ *for each* $\Delta \in \mathbf{F}$ *and* $C \in \mathbf{P}$.

Write $\mathcal{U}(\mathbf{P})$ *for the* ***set of all solutions*** *of* $\mathbf{P}$. $(\mathbf{F}, \sigma) \in \mathcal{U}(\mathbf{P})$ *is* ***more general*** *than* $(\mathbf{F}', \sigma') \in \mathcal{U}(\mathbf{P})$, *written* $(\mathbf{F}, \sigma) \leq (\mathbf{F}', \sigma')$, *if for each* $\Delta' \in \mathbf{F}'$ *there exists* $\Delta \in \mathbf{F}$ *and a v-substitution* θ *such that* $\Delta' \vdash X(\sigma \bullet \theta) \approx_\alpha X\sigma'$ *for all* X *and* $\Delta' \vdash \Delta\theta$. *If there is no* $(\mathbf{F}', \sigma') \in \mathcal{U}(\mathbf{P})$ *such that* $(\mathbf{F}', \sigma') < (\mathbf{F}, \sigma)$ *then* $(\mathbf{F}, \sigma)$ *is a* ***principal*** *or* ***most general solution***.

The unification problem $\{[a \mapsto c]\cdot X \,{}_?{\approx}_?\, c\}$ has principal solutions $(\{\varnothing\}, [X \mapsto a])$ and $(\{\varnothing\}, [X \mapsto c])$. In fact, the unification theory of extended nominal terms is *infinitary*. We give an example after defining complete sets of solutions. Note that solutions of unification problems use collections of contexts, since there may be several independent contexts that solve a constraint, as shown in Example 2.

Definition 6. *Call W a **complete** set of solutions for $\mathbf{P}$ if $W \subseteq \mathcal{U}(\mathbf{P})$; $\forall(\mathbf{F}, \theta) \in W$, $\mathcal{D}om(\theta) \subseteq V(\mathbf{P})$; and $\forall(\mathbf{F}, \sigma) \in \mathcal{U}(\mathbf{P}), \exists(\mathbf{F}', \theta) \in W : (\mathbf{F}', \theta) \leq (\mathbf{F}, \sigma)$. W is a **complete set of most general solutions** if each element is principal.*

The unification problem $\{[c \mapsto f(a,b)]\cdot X \,{}_?{\approx}_?\, f(a, [c \mapsto b]\cdot X)\}$ has an infinite number of principal solutions of the form $(\{\varnothing\}, \sigma_n)$ where $\sigma_n = [X \mapsto f(a, f(a, \ldots, f(a,c)\cdots))]$ and n is the number of occurrences of function symbol f and atom a in $\sigma_n(X)$. In particular, $\sigma_0 = [X \mapsto c]$, $\sigma_1 = [X \mapsto f(a,c)]$ and $\sigma_2 = [X \mapsto f(a, f(a,c))]$.

A **matching constraint** is a unification constraint $s \,{}_?{\approx}_?\, t$ where only variables in s may be instantiated; occurrences of variables in t are seen as constants. We sometimes write $s \,{}_?{\approx} t$ to emphasise that we are dealing with matching, and refer to s as the **pattern** and t as the **matched term**. Matching plays an important role in rewriting: given a set of rewriting rules, the nominal rewriting relation is generated by solving pattern-matching problems as defined below.

Definition 7. *A **matching problem** $\mathbf{P}$ is a set of matching constraints $s_i \,{}_?{\approx} t_i$ such that $(\bigcup_i V(s_i)) \cap (\bigcup_i V(t_i)) = \varnothing$. We denote $\bigcup_i V(s_i)$ by $V_{LHS}(\mathbf{P})$ and $\bigcup_i V(t_i)$ by $V_{RHS}(\mathbf{P})$. A **pattern matching problem** consists of a pair of terms-in-context, written $(\nabla \vdash l) \,{}_?{\approx} (\Delta \vdash t)$ such that $V(\nabla \vdash l) \cap V(\Delta \vdash t) = \varnothing$.*

A solution to a pattern matching problem $(\nabla \vdash l) {}_?{\approx} (\Delta \vdash s)$ is a v-substitution σ, such that there exists $\mathbf{F}$ such that $(\mathbf{F}, \sigma)$ is a solution to $\{l \,{}_?{\approx}\, s\} \cup \nabla$, and $\Delta \vdash \nabla_i$ for some $\nabla_i \in \mathbf{F}$.

Definition 8. *An **extended nominal rewrite rule**, or just **rewrite rule**, is a tuple, written $R = (\nabla \vdash l \to r)$, where ∇ is a freshness context, and l, r are extended nominal terms such that $V(\nabla, r) \subseteq V(l)$. We write $l \to r$ for $\varnothing \vdash l \to r$.*

Example 3.
- $\mathtt{par}(\mathtt{out}(\mathtt{a},\mathtt{b}), \mathtt{in}(\mathtt{a}, [\mathtt{c}]\mathtt{P})) \to [\mathtt{c} \mapsto \mathtt{b}]\cdot\mathtt{P}$ is a rewrite rule , representing communication in the π calculus.
- $\mathtt{a} \# \mathtt{X} \vdash \mathtt{X} \to \mathtt{lam}([\mathtt{a}]\mathtt{app}(\mathtt{X}, \mathtt{a}))$ is the η-expansion rule of the λ-calculus. The β and η reduction rules are:

$$(\beta)\ \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{X}), \mathtt{Y}) \to [\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}$$
$$(\eta)\ \mathtt{a} \# \mathtt{X} \vdash \mathtt{lam}([\mathtt{a}]\mathtt{app}(\mathtt{X}, \mathtt{a})) \to \mathtt{X}$$

  Using standard nominal rules, four additional rules are needed to define explicit substitution (see the Introduction and [13]).
- The higher-order function *map* (see Example 1) is defined by rules:

$$\mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{nil}) \to \mathtt{nil}$$
$$\mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{cons}(\mathtt{H}, \mathtt{T})) \to \mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{T}))$$

To generate the rewrite relation, terms in rewrite rules are considered up to renaming of variables and atoms (metalevel equivariance [24, 13]), denoted t^π.

Definition 9. *A rewrite system $\mathcal{R}$ induces a **rewrite step** $\Delta \vdash s \xrightarrow{R} t$ if there exists $(\nabla \vdash l \to r) \in \mathcal{R}$, $p \in \mathcal{P}os(s)$ and a permutation π such that the pattern-matching problem $(\nabla^\pi \vdash l^\pi)\ {}_?{\approx}\ (\Delta \vdash s|_p)$ has solution θ, and $\Delta \vdash s[r^\pi\theta]_p \approx_\alpha t$:*

$$\frac{\Delta \vdash \{\nabla^\pi\theta, \quad l^\pi\theta \approx_\alpha s|_p, \quad s[r^\pi\theta]_p \approx_\alpha t\}}{\Delta \vdash s \xrightarrow{R} t}\ (\to\mathbf{Rew})$$

*The **(multi-step) rewrite relation** $\Delta \vdash_{\mathcal{R}} s \to t$ is the reflexive, transitive closure of the one-step rewrite relation.*

Example 4. The term-in-context $\vdash \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{lam}([\mathtt{b}]\mathtt{app}(\mathtt{a},\mathtt{b}))),\mathtt{b})$ rewrites to a normal form in one step with the rule (β) (see Example 3), at position ϵ with permutation Id and v-substitution $\theta = [\mathtt{X} \mapsto \mathtt{lam}([\mathtt{b}]\mathtt{app}(\mathtt{a},\mathtt{b}));\mathtt{Y} \mapsto \mathtt{b}]$ as follows,

$$\vdash \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{lam}([\mathtt{b}]\mathtt{app}(\mathtt{a},\mathtt{b}))),\mathtt{b}) \to_{\langle(\beta),\epsilon,\mathsf{Id},\theta\rangle} \mathtt{lam}([\mathtt{c}]\mathtt{app}(\mathtt{b},\mathtt{c}))$$

Capture of the unabstracted atom $\mathtt{b}$ has been avoided by the internal machinery of the extended nominal framework implementing a-substitution. By relegating the semantics of capture-avoiding substitution to the metal-level, where they are managed by our formalism, we have reduced the set of rewrite rules necessary to provide a nominal representation of the rewrite system at hand. The same reduction requires several steps using explicit substitution rules [13].

4 Solving Matching Problems

A sound and complete matching algorithm can be built by converting the set of derivation rules given in Def. 2 into a simplification system. This algorithm can then be used to implement rewriting, and also to check *closedness* of terms and rewrite rules (see [13]). Principal solutions are not unique in general but matching is unitary for a restricted but practically useful class of problems (cf. Theorem 2).

In a matching problem $\mathbf{P} = s_1\ {}_?{\approx} t_, \ldots, s_n\ {}_?{\approx} t_n$, variables on the right-hand side of matching constraints are treated as constants. Hence, without loss of generality, we assume $a \# X$ for any $X \in V_{RHS}(\mathbf{P})$ and $a \in \mathcal{A}$.

Although, initially, the sets of variables in left- and right-hand sides of matching constraints are disjoint, this property is not preserved during the process of solving matching problems (due to variable instantiations). Thus, given a matching problem $\mathbf{P}_0$ to solve, we start by computing the set $V_{RHS}(\mathbf{P}_0)$ of variables that should not be instantiated.

The following auxiliary functions Cap and Ψ are used in the matching algorithm to handle constraints where the pattern is a moderated variable: To solve a constraint of the form $\phi\hat{}\pi\cdot X\ {}_?{\approx}_?\ t$ where $t \neq \phi'\hat{}\pi'\cdot X$, one checks if some subterm $t|_p$ of t is contained in the image of ϕ, that is, $[\pi(a) \mapsto t|_p] \in \phi$. In order to find such position p and subterm $t|_p$, the matching algorithm generates *cap constraints* of the form $(t[a_1 \cdots a_n]_{p_1 \cdots p_n})\phi\ {}_?{\approx}_?\ t$, where $p_i \in \mathcal{P}os(t)$ and $a_i \in \mathcal{D}om(\phi)$, using the function Cap defined below.

Definition 10 (Cap terms). *Let t be a term, $\mathbb{A}$ a finite set of atoms.* $Cap(t,\mathbb{A}) = \{t[a_1 \cdots a_n]_{p_1 \cdots p_n} \mid n \in Nat, a_i \in \mathbb{A}, p_i \in \mathcal{P}os(t), 1 \leq i \leq n\}$.

Thus, $Cap(t,\mathbb{A})$ returns the set of all the terms obtained by replacing subterms of t with atoms from $\mathbb{A}$. Note that $Cap(t,\mathbb{A})$ also includes the term t.

Example 5. $Cap(cons([a \mapsto H]\cdot F, T), \{b, c\}) = \{b, c, cons\ b, cons\ c, cons(b,b), cons(b,c), cons(c,b), cons(c,c), cons(b,T), cons(c,T), cons([a \mapsto b]\cdot F, b), cons([a \mapsto c]\cdot F, b), cons([a \mapsto b]\cdot F, c), cons([a \mapsto c]\cdot F, c), cons([a \mapsto b]\cdot F, T), cons([a \mapsto c]\cdot F, T), cons([a \mapsto H]\cdot F, b), ([a \mapsto H]\cdot F, c), cons([a \mapsto H]\cdot F, T)\}$.

The function Ψ is used to handle constraints of the form $\phi\hat{}\pi\cdot X \ {}_?{\approx}_?\ \phi'\hat{}\pi'\cdot X$ or $a \# \phi\hat{}\pi\cdot X$, i.e., Ψ deals with the premises of rules $(\approx_\alpha \mathbf{X})$ and $(\#\mathbf{X})$ (see Def. 2).

Definition 11 (Function Ψ). *Let s and t be either two moderated variables $\phi\hat{}\pi\cdot X$ and $\phi'\hat{}\pi'\cdot X$, or an atom a and a moderated variable $\phi\hat{}\pi\cdot X$. Let $\mathbf{P}$ be a matching problem, $\mathbb{A}$ a finite set of atoms and b an atom in $\mathbb{A}$. Then, $\Psi(s,t)^{\mathbb{A}} = \Psi'(s,t,\varnothing)^{\mathbb{A}}$ where Ψ' computes a set of problems (i.e., a collection of sets of constraints) as follows:* $\Psi'(s,t,\mathbf{P})^{\mathbb{A}} \triangleq$

$$\begin{cases} \bullet\ \{\mathbf{P}\} & \text{if } \mathbb{A} = \varnothing \\ \bullet\ \Psi'(s,t,\mathbf{P} \cup \{s \# \phi(b)\})^{\mathbb{A}\setminus\{b\}} \cup \Psi'(s,t,\mathbf{P} \cup \{\pi^{-1}(b) \# X\})^{\mathbb{A}\setminus\{b\}} & \text{if } s = a, t = \phi\hat{}\pi\cdot X \\ \bullet\ \Psi'(s,t,\mathbf{P} \cup \{\phi(\pi(b)) \ {}_?{\approx}_?\ \phi'(\pi'(b))\})^{\mathbb{A}\setminus\{b\}} \cup \Psi'(s,t,\mathbf{P} \cup \{b \# X\})^{\mathbb{A}\setminus\{b\}} & \text{if } s = \phi\hat{}\pi\cdot X, t = \phi'\hat{}\pi'\cdot X \end{cases}$$

Ψ' deals with one atom from the given finite set $\mathbb{A}$ in each recursive call (thus ensuring termination); the order in which elements of $\mathbb{A}$ are considered is irrelevant since $\Psi'(s,t,\mathbf{P})^{\mathbb{A}}$ is a collection of sets. Hence Ψ' is indeed a function.

Freshness constraints of form $a \# a$ are **inconsistent**. A matching constraint $s \ {}_?{\approx}_?\ t$ is **clashing** when s, t have different term constructors at the root except if s is a moderated variable $\phi\hat{}\pi\cdot X$ and $X \notin V_{RHS}(\mathbf{P})$. For example, if $V_{RHS}(\mathbf{P}) = \{Y\}$ then $a \ {}_?{\approx}_?\ (a\ b)\cdot Y$, $f\ a \ {}_?{\approx}_?\ g\ a$ and $[a \mapsto b]\cdot Y \ {}_?{\approx}_?\ [b]a$ are clashing but $[a \mapsto b]\cdot X \ {}_?{\approx}_?\ f(c,b)$ and $[a \mapsto b]\cdot Y \ {}_?{\approx}_?\ [b \mapsto a]\cdot Y$ are not. Clashing and inconsistent constraints are not derivable; failure rules $(\bot)$ will be specified to deal with them.

Definition 12 (Matching steps). *Let $\mathbf{P}_0$ be a matching problem and $\mathbb{X} = V_{RHS}(\mathbf{P}_0)$. $\mathcal{P}, \mathcal{Q}$ denote sets of pairs $(\mathbf{P}, \theta)$, where $\mathbf{P}$ is a unification problem and θ a v-substitution. Write $\mathcal{P} \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} \mathcal{Q}$ (resp. $\mathcal{P} \Longrightarrow_{\#} \mathcal{Q}$), if $\mathcal{Q}$ is obtained from $\mathcal{P}$ by application of one matching (resp. freshness) reduction rule below. As usual, $\Longrightarrow^*_{?\approx}$ (resp. $\Longrightarrow^*_{\#}$) denotes reflexive transitive closure; arrow subindices are omitted if there is no ambiguity.*

$$\begin{array}{lll}
({}_?{\approx}\bot)^1 & (\{s \ {}_?{\approx}_?\ t\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} \varnothing & \text{if clashing} \\
({}_?{\approx}\equiv) & (\{t \ {}_?{\approx}_?\ t\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} (\mathbf{P}, \theta) & \\
({}_?{\approx}\mathbf{f}) & (\{f s \ {}_?{\approx}_?\ f t\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} (\{s \ {}_?{\approx}_?\ t\} \cup \mathbf{P}, \theta) & \\
({}_?{\approx}[\mathbf{a}]) & (\{[a]s \ {}_?{\approx}_?\ [a]t\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} (\{s \ {}_?{\approx}_?\ t\} \cup \mathbf{P}, \theta) & \\
({}_?{\approx}[\mathbf{b}]) & (\{[a]s \ {}_?{\approx}_?\ [b]t\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} (\{(b\ a)\cdot s \ {}_?{\approx}_?\ t, b \# s\} \cup \mathbf{P}, \theta) & \\
({}_?{\approx}\mathbf{tupl}) & (\{(s_1, \ldots, s_n) \ {}_?{\approx}_?\ (t_1, \ldots, t_n)\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} (\{s_1 \ {}_?{\approx}_?\ t_1, \ldots, s_n \ {}_?{\approx}_?\ t_n\} \cup \mathbf{P}, \theta) &
\end{array}$$

$$(?\approx\mathbf{X})^1 \quad (\{\phi\hat{}\pi\cdot X \;?\approx_?\; \phi'\hat{}\pi'\cdot X\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} \bigcup_{\mathbf{P}' \in \Psi(\phi\hat{}\pi\cdot X, \phi'\hat{}\pi'\cdot X)^{\mathbb{A}}} \{(\mathbf{P}' \cup \mathbf{P}, \theta)\}$$
$$\text{where } \mathbb{A} = (\mathcal{S}upportP(\pi, \pi') \cup \mathcal{D}omP(\phi, \phi'))$$

$$(?\approx\mathbf{Inst})^1 \quad (\{\phi\hat{}\pi\cdot X \;?\approx_?\; t\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} \bigcup_{s \in Cap(t, \mathcal{D}om(\phi))} \{(\{s(\phi\theta') \;?\approx_?\; t\} \cup \mathbf{P}\theta', \theta \bullet \theta')\}$$
$$\text{if } t \neq \phi'\hat{}\pi'\cdot Y \;\; (Y \in \mathcal{X}),\;\; (X \notin \mathbb{X}),\;\; \theta' = [X \mapsto \pi^{-1}\cdot s]$$

$$(?\approx\mathbf{XY})^1 \quad (\{\phi\hat{}\pi\cdot X \;?\approx_?\; \phi'\hat{}\pi'\cdot Y\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} \bigcup_{s \in (Cap(\phi'\hat{}\pi'\cdot Y, \mathcal{D}om(\phi)) \cup \{\pi'\cdot Y\})} \{(\{s(\phi\theta') \;?\approx_?\; \phi'\hat{}\pi'\cdot Y\} \cup \mathbf{P}\theta', \theta \bullet \theta')\}$$
$$\text{if } (X \notin \mathbb{X}) \text{ and } \theta' = [X \mapsto \pi^{-1}\cdot s]$$

$$(\#\bot) \quad \{a \# a\} \cup \mathbf{P} \Longrightarrow_{\#} \bot$$
$$(\#\mathbf{ab}) \quad \{a \# b\} \cup \mathbf{P} \Longrightarrow_{\#} \mathbf{P}$$
$$(\#[\mathbf{a}]) \quad \{a \# [a]s\} \cup \mathbf{P} \Longrightarrow_{\#} \mathbf{P}$$
$$(\#[\mathbf{b}]) \quad \{a \# [b]s\} \cup \mathbf{P} \Longrightarrow_{\#} \{a \# s\} \cup \mathbf{P}$$
$$(\#\mathbf{f}) \quad \{a \# fs\} \cup \mathbf{P} \Longrightarrow_{\#} \{a \# s\} \cup \mathbf{P}$$
$$(\#\mathbf{tupl}) \quad \{a \# (s_1, \ldots, s_n)\} \cup \mathbf{P} \Longrightarrow_{\#} \{a \# s_1, \ldots, a \# s_n\} \cup \mathbf{P}$$
$$(\#\mathbf{X})^1 \quad \{a \# \phi\hat{}\pi\cdot X\} \cup \mathbf{P} \Longrightarrow_{\#} \bigcup_{\mathbf{P}' \in \Psi(a, \phi\hat{}\pi\cdot X)^{\mathbb{A}}} \{\mathbf{P}' \cup \mathbf{P}\} \quad (\text{where } \mathbb{A} = \mathcal{D}om(\phi) \cup \{a\})$$
$$\text{if } \phi \neq \mathsf{Id} \wedge \pi \neq \mathsf{Id}$$

Rule $(?\approx\equiv)$ has priority; it is an optimisation to reduce trivial matching constraints in one step, subsuming rule $(\approx_\alpha\mathbf{a})$ (Def. 2). The right-hand side of rule $(?\approx\bot)$ is the empty set since this pair cannot produce solutions (but other pairs in the problem could, so we do not use $\bot$). Rules $(?\approx\mathbf{Inst})$ and $(?\approx\mathbf{XY})$ are **instantiating rules**. Note that the matching steps in Def. 12 provide an algorithmic presentation of Def. 2, where instantiating rules have been added and the symbol $\approx_\alpha$ has been replaced by $?\approx_?$ to represent the constraints to be solved.

Termination of the simplification process follows from the fact that the instantiating rules decrease the number of variables in the problem. For any other rule, an interpretation based on the multiset of sizes of the constraints in the problem can be shown to be strictly decreasing using the multiset extension of the standard ordering on natural numbers, $\leq_{mul}$. Confluence under the imposed strategy then follows by Newman's Lemma, since there are only trivial overlaps. Hence normal forms are unique.

Remark 1 (Matching algorithm). The algorithm has *two phases.*
Input: Assume $\mathbf{P}_0$ is the given matching problem, where $\mathbb{X} = V_{RHS}(\mathbf{P}_0)$.
Phase 1 ($\Longrightarrow_{?\approx}$-Normalisation): $\{(\mathbf{P}_0, \mathsf{Id})\} \overset{\mathbb{X}}{\Longrightarrow}^*_{?\approx} \langle\mathbf{P}_0\rangle_{\mathtt{nf}_{?\approx}}$ where $\langle\mathbf{P}_0\rangle_{\mathtt{nf}_{?\approx}}$ is the normal form of $\{(\mathbf{P}_0, \mathsf{Id})\}$ by application of $\Longrightarrow_{?\approx}$.
Phase 2 ($\Longrightarrow_{\#}$-Normalisation): $\forall(\mathbf{P}_i, \theta_i) \in \langle\mathbf{P}_0\rangle_{\mathtt{nf}_{?\approx}}$, compute $\{\mathbf{P}_i\} \Longrightarrow^*_{\#} \langle\mathbf{P}_i\rangle_{\mathtt{nf}_\#}$ where $\langle\mathbf{P}_i\rangle_{\mathtt{nf}_\#}$ is the normal form of the set of freshness constraints $\mathbf{P}_i$ by application of $\Longrightarrow_{\#}$.
Output: $\langle\mathbf{P}_0\rangle_{out} = \{(\langle\mathbf{P}_i\rangle_{\mathtt{nf}_\#} \setminus \bot, \theta_i) \mid (\mathbf{P}_i, \theta_i) \in \langle\mathbf{P}_0\rangle_{\mathtt{nf}_{?\approx}}, \langle\mathbf{P}_i\rangle_{\mathtt{nf}_\#} \neq \bot\}$.

Informally, **Phase 1** reduces the matching problem until no matching constraints are left, resolving into a set of pairs $(\{C_{ij}\}, \theta_i)(i, j \in Nat)$ where each

[1] In this rule, the right-hand side is a set; we assume a flattening step is performed after each application of the rule (to avoid nested sets).

C_{ij} is a (possibly empty) set of freshness constraints and θ_i a v-substitution. Then, **Phase 2** reduces each C_{ij} into freshness contexts C'_{ij}, discarding along the way any set C_{ij} containing inconsistent freshness constraints. Finally, the remaining pairs (C'_{ij}, θ_i) in the set are solutions to the initial matching problem $\mathbf{P}_0$. If no pairs are left, i.e., all sets of freshness constraints have been discarded, then the matching problem is unsolvable.

Example 6. The matching problem $\mathbf{P} = (\{[a \mapsto Y]\cdot X \stackrel{?}{\approx} [a \mapsto b]\cdot Z\})$ has principal solutions $(\{a \# Z\}, [X \mapsto Z])$, $(\varnothing, [X \mapsto Z; Y \mapsto b])$, $(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])$, $(\{a \# Z\}, [X \mapsto a; Y \mapsto Z])$, $(\varnothing, [X \mapsto a; Y \mapsto [a \mapsto b]\cdot Z])$ computed by the algorithm as follows. Below, the affected parts of each reduction are highlighted and outer brackets in singleton collections of freshness contexts are omitted for readability. Here $V_{RHS}(\mathbf{P}) = \{Z\}$.

$$
\begin{aligned}
& \{\ (\{[a \mapsto Y]\cdot X \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, \mathsf{Id})\ \} \\
\overset{X \notin \{Z\}}{\Longrightarrow}_{(?\approx\mathbf{XY})}\ & \{(\{Y \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, \mathsf{Id} \bullet [X \mapsto a])\} \\
& \cup\{(\{[a \mapsto Y]\cdot Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, \mathsf{Id} \bullet [X \mapsto Z])\} \\
& \cup\{\ (\{[a \mapsto b]\cdot Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, \mathsf{Id} \bullet [X \mapsto [a \mapsto b]\cdot Z])\ \} \\
& \textit{where } Cap([a \mapsto b]\cdot Z, \{a\}) = \{a, Z, [a \mapsto b]\cdot Z\} \\
\Longrightarrow_{(?\approx\equiv)}\ & \{\ (\{Y \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto a])\ \} \cup \\
& \{(\{[a \mapsto Y]\cdot Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto Z])\} \cup \{(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])\} \\
\overset{Y \notin \{Z\}}{\Longrightarrow}_{(?\approx\mathbf{XY})}\ & \{\ (\{[a \mapsto b]\cdot Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto a] \bullet [Y \mapsto [a \mapsto b]\cdot Z])\ \} \\
& \cup\{(\{Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto a] \bullet [Y \mapsto Z])\} \cup \\
& \{(\{[a \mapsto Y]\cdot Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto Z])\} \cup \{(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])\} \\
& \textit{where } Cap([a \mapsto b]\cdot Z, \varnothing) = \{[a \mapsto b]\cdot Z\} \\
\Longrightarrow_{(?\approx\equiv)}\ & \{(\varnothing, [X \mapsto a] \bullet [Y \mapsto [a \mapsto b]\cdot Z])\} \\
& \cup\{\ (\{Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto a] \bullet [Y \mapsto Z])\ \} \cup \\
& \{(\{[a \mapsto Y]\cdot Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto Z])\} \cup \{(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])\} \\
\Longrightarrow_{(?\approx\mathbf{X})}\ & \{(\varnothing, [X \mapsto a] \bullet [Y \mapsto [a \mapsto b]\cdot Z])\} \\
& \cup\{(\{a \# Z\}, [X \mapsto a] \bullet [Y \mapsto Z])\} \\
& \cup\{\ (\{a \stackrel{?}{\approx} b\}, [X \mapsto a] \bullet [Y \mapsto Z])\ \} \\
& \cup\{\ (\{[a \mapsto Y]\cdot Z \stackrel{?}{\approx} [a \mapsto b]\cdot Z\}, [X \mapsto Z])\ \} \\
& \cup\{(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])\} \\
& \textit{where } \Psi(Z, [a \mapsto b]\cdot Z)^{\{a\}} = \{\{a \stackrel{?}{\approx} b\}, \{a \# Z\}\} \\
\Longrightarrow_{(?\approx\bot)}\ & \Longrightarrow_{(?\approx\mathbf{X})} \quad \{(\varnothing, [X \mapsto a] \bullet [Y \mapsto [a \mapsto b]\cdot Z])\} \\
& \cup\{(\{a \# Z\}, [X \mapsto a] \bullet [Y \mapsto Z])\} \cup \{(\{a \# Z\}, [X \mapsto Z])\} \\
& \cup\{\ (\{Y \stackrel{?}{\approx} b\}, [X \mapsto Z])\ \} \cup \{(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])\} \\
& \textit{where } \Psi([a \mapsto Y]\cdot Z, [a \mapsto b]\cdot Z)^{\{a\}} = \{\{a \# Z\}, \{Y \stackrel{?}{\approx} b\}\} \\
\overset{Y \notin \{Z\}}{\Longrightarrow}_{(?\approx\mathbf{Inst})}\ & \Longrightarrow_{(?\approx\equiv)} \quad \{(\varnothing, [X \mapsto a] \bullet [Y \mapsto [a \mapsto b]\cdot Z])\} \\
& \cup\{(\{a \# Z\}, [X \mapsto a] \bullet [Y \mapsto Z])\} \cup \{(\{a \# Z\}, [X \mapsto Z])\} \\
& \cup\{(\varnothing, [X \mapsto Z] \bullet [Y \mapsto b])\} \cup \{(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])\}.
\end{aligned}
$$

Phase 2 is trivial and thus omitted.

As a consequence of the termination and confluence properties, the relation $\Longrightarrow$ defines a function from matching problems to their unique normal form. Write $\langle \mathbf{P} \rangle_{out}$ for the normal form of $\{(\mathbf{P}, \mathsf{Id})\}$. $\langle \mathbf{P} \rangle_{out}$ may contain solutions $(\mathbf{F}, \sigma), (\mathbf{F}', \sigma')$ with α-equivalent substitutions but different collections of freshness contexts. For instance, $(\varnothing, [X \mapsto a; Y \mapsto [a \mapsto b]\cdot Z])$ and $(\{a \# Z\}, [X \mapsto a; Y \mapsto Z])$ in Example 6 could be merged as $(\varnothing, [X \mapsto a; Y \mapsto [a \mapsto b]\cdot Z])$.

Definition 13 (Merging solutions). *Let W be a set of solutions, such that there are two different elements $(\mathbf{F}, \sigma)$ and $(\mathbf{F}', \sigma')$ in W satisfying $\forall \Delta \in \mathbf{F}.\Delta \vdash \sigma \approx_\alpha \sigma'$. This pair of solutions can be replaced with a single solution as follows:*

$$([\mathbf{W_1}]) \qquad (\mathbf{F}, \sigma), (\mathbf{F}', \sigma') \quad \Longrightarrow_{[W]} \quad (\mathbf{F}' \cup \mathbf{F}, \sigma')$$

Further, if $(\mathbf{F}, \sigma)$ contains the empty set as one of the freshness contexts in $\mathbf{F}$, then any other freshness context in $\mathbf{F}$ is redundant and can be discarded:

$$([\mathbf{W_2}]) \quad (\mathbf{F}, \sigma) \quad \Longrightarrow_{[W]} \quad (\{\varnothing\}, \sigma) \quad \text{if } \mathbf{F} \neq \{\varnothing\},\ \varnothing \in \mathbf{F}$$

Write $[W]$ for the normal form of W by the rules above. $\langle \mathbf{P} \rangle_{sol}$ denotes the normal form by $([\mathbf{W_1}])$ and $([\mathbf{W_2}])$ of $\langle \mathbf{P} \rangle_{out}$, that is: $\langle \mathbf{P} \rangle_{sol} = [\langle \mathbf{P} \rangle_{out}]$.

Example 7 (Merging solutions). By application of rules $[W_1]$ and $[W_2]$ to the solution set W from Example 6, solution $(\varnothing, [X \mapsto a; Y \mapsto [a \mapsto b]\cdot Z])$ replaces in W the pair $(\varnothing, [X \mapsto a; Y \mapsto [a \mapsto b]\cdot Z])$, $(\{a \# Z\}, [X \mapsto a; Y \mapsto Z])$. Similarly, $(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])$ replaces the pair $(\varnothing, [X \mapsto [a \mapsto b]\cdot Z])$, $(\{a \# Z\}, [X \mapsto Z])$.

Theorem 1 (Soundness and completeness). $\langle \mathbf{P} \rangle_{sol} \subseteq \mathcal{U}(\mathbf{P})$ *(soundness);*
$\forall (\mathbf{F}, \sigma) \in \mathcal{U}(\mathbf{P})$, $\exists (\mathbf{F}_i, \theta_i) \in \langle \mathbf{P} \rangle_{sol}$ *such that* $(\mathbf{F}_i, \theta_i) \leq (\mathbf{F}, \sigma)$ *(completeness).*

5 Unitary Matching for Simple Problems

When using matching to generate, for instance, rewrite steps for a given nominal rewriting rule, it is useful to have a unique most general matching solution. Below we characterise a class of matching constraints, which we call **simple**, for which matching is unitary. The idea is to require each variable symbol in a pattern to have at least one occurrence with trivial suspended a-substitution (Id) and not in a suspension (see below). Constraints whose pattern is a moderated variable with non-trivial a-substitutions will be **postponed**.

Moderated variables occurring in suspended a-substitutions will be called **suspended (variable) occurrences**, and the others will be called **fixed (variable) occurrences**. For instance, in the term $([a \mapsto Z]\cdot X, [a \mapsto b]\cdot Y)$, both $[a \mapsto Z]\cdot X$ and $[a \mapsto b]\cdot Y$, are fixed, but Z is a suspended occurrence since it occurs in the image of the a-substitution suspended over X. Write $V_f(t)$ for the subset of $V(t)$ such that each variable *has at least one fixed occurrence with trivial a-substitutions.* The set $V_f(t)$ will play an important role in the characterisation of unitary matching problems.

Definition 14. *A term s is **simple** if $V(s) \subseteq V_f(s)$, that is, for each variable $X \in V(s)$ there is one or more fixed occurrences of the form $\mathsf{Id}\hat{}\pi{\cdot}X$.*[2]

*A **simple matching constraint** is a matching constraint $s \,{}_?{\approx}\, t$ such that s is a simple term and $V(s) \cap V(t) = \varnothing$. A **simple matching problem** is a problem as specified in Def. 7 where $(\dots, s_i, \dots) \,{}_?{\approx}\, (\dots, t_i, \dots)$ is simple.* [3]

Example 8. The constraints $\mathtt{cons}([a \mapsto Y]{\cdot}X, \mathtt{map}([b]X, Y)) \,{}_?{\approx}\, \mathtt{cons}(H, \mathtt{nil})$ and $\mathtt{map}([a]X, \mathtt{cons}(Y, \mathtt{nil})) \,{}_?{\approx}\, \mathtt{map}([a \mapsto H]{\cdot}F, \mathtt{cons}(H, \mathtt{nil}))$ are simple but the constraint $\mathtt{map}([a \mapsto Y]{\cdot}X, X) \,{}_?{\approx}\, \mathtt{map}([a \mapsto \mathtt{nil}]{\cdot}F, F)$ is not; the latter does not have a simple pattern term, there is no fixed occurrence of Y.

Given a simple matching problem $\mathbf{P}$, postponed constraints (of the form $\phi\hat{}\pi{\cdot}X \,{}_?{\approx}\, t$ where $t \neq \phi'\hat{}\pi'{\cdot}X$, $\phi \neq \mathsf{Id}$ and $X \notin \{X \mid s \,{}_?{\approx}\, t \in \mathbf{P}, X \in V(t)\}$) are delayed until an instantiation for X is readily available. The definition of simple constraint (Def. 14) ensures such instantiation exists. The matching rule $({}_?{\approx}\mathbf{XY})$ is not included in the simple-matching algorithm and rule $({}_?{\approx}\mathbf{Inst})$ is adapted following the standard instantiating rule (see [28, Fig. 3]) as follows.

Definition 15 (Simple-matching algorithm). *Let $\mathbf{P}$ be a simple matching problem, $\mathbb{X} = V_{RHS}(\mathbf{P})$ and assume $X \notin \mathbb{X}$. Take the rule set of Def. 12, discard rule $({}_?{\approx}\mathbf{XY})$ and replace rule $({}_?{\approx}\mathbf{Inst})$ with:*

$$({}_?{\approx}\sigma)\ (\{\pi{\cdot}X \,{}_?{\approx}\, t\} \cup \mathbf{P}, \theta) \overset{\mathbb{X}}{\Longrightarrow}_{?\approx} (\mathbf{P}[X \mapsto \pi^{-1}{\cdot}t], \theta \bullet [X \mapsto \pi^{-1}{\cdot}t])$$

*The **simple-matching algorithm** follows the two-phase reduction strategy described in Remark 1, using the modified rule set where rule $({}_?{\approx}\sigma)$ has the highest priority along with rule $({}_?{\approx}{\equiv})$, rule $({}_?{\approx}\mathbf{X})$ has the lowest priority and all other rules have equal priority. Let $\langle\mathbf{P}\rangle_{\mathtt{nf}_{?\approx}}$ be the normal form of $\mathbf{P}$ with respect to the set of updated rules.*

The priority imposed on rule $({}_?{\approx}\sigma)$ forces the generation of v-substitutions as soon as possible, whilst by giving lowest precedence to the rule $({}_?{\approx}\mathbf{X})$, we ensure it is simply checking α-equality (as specified by $({\approx_\alpha}\mathbf{X})$) since no variables are left to be instantiated. As a result, each distinct solution $(\mathbf{F}, \sigma)$ from the solution set W shares the same unifier, σ, and by application of the merging rule to W in the final part of the algorithm, the solution set is reduced to $[W] = \{(\bigcup \mathbf{F}, \sigma)\}$. We formalise this claim in Theorem 2. Write $\mathtt{Match}(\mathbf{P}, V_{RHS}(\mathbf{P}))$ for the normal form of the matching problem $\mathbf{P}$ by the simple-matching algorithm (Def. 15). Then, $\langle\mathbf{P}\rangle_{sol_{?\approx}}$ is the result of applying function $[\cdot]$ from Def. 13 to $\mathtt{Match}(\mathbf{P}, V_{RHS}(\mathbf{P}))$. The following theorem is the main result of this section.

Theorem 2 (Normal form of a simple problem). *Given a simple matching problem $\mathbf{P}$, either $\langle\mathbf{P}\rangle_{sol_{?\approx}} = [\mathtt{Match}(\mathbf{P}, V_{RHS}(\mathbf{P}))] = \{(\mathbf{F}, \theta)\}$ and $(\mathbf{F}, \theta)$ is a solution for $\mathbf{P}$, or $\langle\mathbf{P}\rangle_{sol_{?\approx}} = \varnothing$ and $\mathbf{P}$ has no solution.*

[2] This means that each variable has an occurrence that does not involve a-substitution.

[3] We use a constraint $(\dots, s_i, \dots) \,{}_?{\approx}\, (\dots, t_i, \dots)$ in order to ensure that all the variables that can be instantiated have an occurrence that does not involve a-substitution somewhere in the problem.

Example 9. The rewriting rules in Example 3 have simple terms as patterns and the rewrite relation generated uses only simple matching: indeed, all the terms used in left-hand sides of rewrite rules are standard nominal terms (without atom substitutions), only the matched terms may have a-substitutions.

A-substitutions are used in the right-hand side of rules in Example 3 to implement function application in a direct way (avoiding the introduction of an additional set of rewrite rules to define non-capturing atom substitution as in standard nominal rewriting systems).

6 Undecidability of Extended Nominal Unification

To prove the undecidability of extended nominal unification, we encode Hilbert's tenth problem, proved undecidable in [22]. The main idea is to build unification problems for which *ground unifiers* simulate addition or multiplication. Then, one can represent Diophantine equations. To simplify the encoding, we consider a restricted language.

Definition 16 (Terms in L). L-***terms*** *are generated from a triple* $(\mathcal{A}, \mathcal{X}, \mathcal{F}_L)$ *of pairwise disjoint sets, where* $\mathcal{F}_L$ *is empty and* $\mathcal{X}, \mathcal{A}$ *are countable sets of variables and atoms respectively (as described in Section 2), using the grammar given in Def. 1 without abstraction terms.*

Our representation of natural numbers is inspired by Goldfarb numbers [18], which are themselves inspired by Church numerals. In L, the natural number n is written: $\overline{n}_a c = (a, (a, \dots (a, c)))$ with n occurrences of a and a single occurrence of c, where $a, c \in \mathcal{A}$. L-terms of this form, which we call **L-Goldfarb numbers**, are exactly those that solve extended nominal unification problems of the form

$$\{(a, [c \mapsto a]{\cdot}F) \mathrel{?}\approx_? [c \mapsto (a, a)]{\cdot}F\}. \tag{1}$$

Example 10 (L-Goldfarb numbers). The number 0 is represented as $\overline{0}_a c$, that is, c; the number 1 is represented as $\overline{1}_a c = (a, c)$; 3 is represented as $\overline{3}_a c = (a, (a, (a, c)))$. The term $\overline{2}_a(\overline{1}_a a)$ is in L but is not an L-Goldfarb number (it does not solve Eqn. 1). Note also that, $\overline{2}_a(\overline{1}_a a) = (a, (a, (a, a))) = \overline{2+1}_a a$.

To simulate addition, we adapt Church's λ-term $add = \lambda n.\lambda m.\lambda x.n(m(x))$: we use a constraint $[c \mapsto X_i]{\cdot}X_j \mathrel{?}\approx_? X_k$. To simulate multiplication we use nested a-substitutions. Undecidability of extended nominal unification follows from Lem. 1 & 2.

Lemma 1 (Addition). *For all* $m, n, p \geq 0$*, there exists a ground unifier* θ *for the unification problem* $\{[c \mapsto X_i]{\cdot}X_j \mathrel{?}\approx_? X_k\}$ *such that* $\{[X_i \mapsto \overline{n}_a c; X_j \mapsto \overline{m}_a c; X_k \mapsto \overline{p}_a c]\} \subseteq \theta$ *if and only if* $p = m + n$.

Lemma 2 (Multiplication). *Let* $\mathbf{P}^{\times} = \{s_1 \mathrel{?}\approx_? s_2, s_3 \mathrel{?}\approx_? s_4\}$ *where*
$s_1 = [c_1 \mapsto a; c_2 \mapsto b; c_3 \mapsto (([c \mapsto a]{\cdot}X_k, [c \mapsto b]{\cdot}X_j), a)]{\cdot}G$,
$s_2 = ((a, b), [c_1 \mapsto [c \mapsto a]{\cdot}X_i; c_2 \mapsto \overline{1}_a b; c_3 \mapsto a]{\cdot}G)$,

$s_3 = [c_1 \mapsto b; c_2 \mapsto a; c_3 \mapsto (([c \mapsto b]\cdot X_k, [c \mapsto a]\cdot X_j), b)]\cdot G$,
$s_4 = ((b, a), [c_1 \mapsto [c \mapsto b]\cdot X_i; c_2 \mapsto \overline{1}_a a; c_3 \mapsto b]\cdot G)$.

For all $m, n, p \geq 0$, there is a ground unifier θ for $\mathbf{P}^{\times}$ such that $\sigma = [X_i \mapsto \overline{m}_a c; X_j \mapsto \overline{n}_a c; X_k \mapsto \overline{p}_a c]$ and $\sigma \subset \theta$ if and only if $p = m \times n$.

Theorem 3. *There is an effective reduction of Hilbert's tenth problem to nominal unification of* L*-terms. Therefore unification of extended terms is undecidable.*

7 Conclusion

The matching algorithm provided in this paper induces a notion of rewriting that avoids the need to introduce extra rules to encode non-capturing substitutions. In future work, we will analyse the relationship between higher-order matching/unification and the corresponding problems in our language. The study of the complexity of the algorithms and the development of efficient implementations using graph representations of terms will also be subject of future research.

Acknowledgements We thank James Cheney, Elliot Fairweather and Jordi Levy for helpful comments and useful pointers.

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, New York, NY, USA (1998)
2. Byrd, W., Friedman, D.: α-kanren: A fresh name in nominal logic programming. In: Proceedings of the 2007 Workshop on Scheme and Functional Programming. pp. 79–90. Université Laval Technical Report DIUL-RT-0701 (2007)
3. Calvès, C.: Unifying Nominal Unification. In: RTA-13. LIPIcs, vol. 21, pp. p. 143–157 (2013), http://drops.dagstuhl.de/opus/volltexte/2013/4059
4. Calvès, C., Fernández, M.: Nominal matching and alpha-equivalence. In: WoLLIC 2008. pp. p. 111–122 (2008)
5. Calvès, C., Fernández, M.: A polynomial nominal unification algorithm. Theor. Comput. Sci. **403**(2-3), p. 285–306 (2008)
6. Calvès, C., Fernández, M.: Matching and alpha-equivalence check for nominal terms. Journal of Comput. and Sys. Sci. (2009), special issue: Selected papers from WOLLIC 2008
7. Calvès, C., Fernández, M.: The first-order nominal link. In: LOPSTR 2010. pp. p. 234–248 (2010)
8. Cheney, J., Urban, C.: αProlog: A logic programming language with names, binding and α-equivalence. In: Logic Programming, pp. 269–283. Springer Berlin Heidelberg (2004)
9. Cheney, J.: The complexity of equivariant unification. In: Automata, Languages and Programming, Proceedings of the 31st Int. Colloquium, ICALP 2004. Lecture Notes in Computer Science, vol. 3142. Springer (2004)
10. Clouston, R.A., Pitts, A.M.: Nominal equational logic. In: Cardelli, L., Fiore, M., Winskel, G. (eds.) Computation, Meaning and Logic. Articles dedicated to Gordon Plotkin, ENTCS, vol. 1496. Elsevier (2007)

11. Dowek, G., Gabbay, M., Mulligan, D.: Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. IGPL-10 **18**(6), p. 769–822 (2010)
12. Fairweather, E., Fernández, M., Szasz, N., Tasistro, A.: Dependent Types for Nominal Terms with Atom Substitutions. In: TLCA-15. LIPIcs, vol. 38, pp. p. 180–195 (2015), http://drops.dagstuhl.de/opus/volltexte/2015/5163
13. Fernández, M., Gabbay, M.: Nominal rewriting. Inf. Comput. **205**(6), p. 917–965 (2007), http://dx.doi.org/10.1016/j.ic.2006.12.002
14. Fernández, M., Gabbay, M.J.: Closed nominal rewriting and efficiently computable nominal algebra equality. In: Proceedings 5th Int. Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010. pp. 37–51 (2010). https://doi.org/10.4204/EPTCS.34.5
15. Gabbay, M.J., Mathijssen, A.: Capture-avoiding substitution as a nominal algebra. Formal Aspects of Comp. **20**(4-5), p. 451–479 (2008), http://dx.doi.org/10.1007/s00165-007-0056-1
16. Gabbay, M.J., Mathijssen, A.: Nominal universal algebra: Equational logic with names and binding. Journal of Log. and Comp. (2009)
17. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing **13**(3–5), 341–363 (2001)
18. Goldfarb, W.D.: The undecidability of the second-order unification problem. Theo. Comp. Sci. **13**(2), p. 225–230 (1981)
19. Kumar, R., Norrish, M.: (nominal) unification by recursive descent with triangular substitutions. In: IITP 2010. pp. p. 51–66 (2010)
20. Levy, J., Veanes, M.: On the undecidability of second-order unification. Inf. Comput. **159**(1-2), p. 125–150 (2000), https://doi.org/10.1006/inco.2000.2877
21. Levy, J., Villaret, M.: An efficient nominal unification algorithm. In: RTA-2010. pp. p. 209–226 (2010), http://dx.doi.org/10.4230/LIPIcs.RTA.2010.209
22. Matiyasevich, Y.V.: Enumerable sets are Diophantine (in Russian). Soviet Math. Doklady **191**(2), p. 279–282 (1970)
23. Near, J.P., Byrd, W.E., Friedman, D.P.: alpha-leanTAP: A declarative theorem prover for first-order classical logic. In: Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings. pp. 238–252 (2008), https://doi.org/10.1007/978-3-540-89982-2_26
24. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. **186**(2), p. 165–193 (2003), https://doi.org/10.1016/S0890-5401(03)00138-X
25. Pitts, A.M., Gabbay, M.: A metalanguage for programming with bound names modulo renaming. In: MPC 2000. pp. p. 230–255 (2000)
26. Shinwell, M.R., Pitts, A.M., Gabbay, M.J.: FreshML: programming with binders made simple. SIGPLAN-03 **38**(9), p. 263–274 (2003)
27. Suzuki, T., Kikuchi, K., Aoto, T., Toyama, Y.: Confluence of orthogonal nominal rewriting systems revisited. In: Fernández, M. (ed.) 26th Int. Conference on Rewriting Techniques and Applications, RTA 2015, June 29 to July 1, 2015, Warsaw, Poland. LIPIcs, vol. 36, pp. 301–317. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015), https://doi.org/10.4230/LIPIcs.RTA.2015.301
28. Urban, C., Pitts, A.M., Gabbay, M.: Nominal unification. Theor. Comput. Sci. **323**(1-3), p. 473–497 (2004)